STAT 243 Final Group Project: Adaptive Rejection Sampling

Jinze Gu, Havard Kvamme, Reid Stevens and Yang Wang

December 13, 2013

The R file was submitted by [name].

Adaptive Rejection Sampling (ARS) is a technique for generating random samples from a distribution with a log-concave density function developed by Gilks and Wild (1992). Given a set of starting values, ARS constructs piecewise-linear upper bounds using tangent lines to the log density function and piecewise-linear lower bounds using secant lines to the density function. The bounds are used to determine whether sampled points are accepted or rejected. These bounds are improved during the sampling process using the chosen values, which reduces the probability of needing to evaluate the density function.

Our 'ars' function performs sampling from a log-concave probability density function using ARS. The inputs to the 'ars' function are the log-concave density function of interest ('f'), the number of points to sample from the density function ('n'), and the initial set of abscissae ('x_init'). The left and right bounds on the domain of the function ('left_bound' and 'right_bound') are assumed to be $\pm\infty$ unless specified. Extra arguments can also be passed to the function through the 'ars' function.

The 'ars' function is composed of seven subroutines: 'make_z', 'make_upper_bound', 'make_lower_bound', 'sample_upper_bound', 'filter', 'update_sample', and 'update_x'. The initial step takes the set of starting values ('x_init') and evaluates the log of the density function at these points (creating the vector 'hx'). The derivative of the the log of the density function is estimated at the initial values with the auxiliary function 'numericDeriv' (creating the vector 'hpx'). If the density function is unbounded, the function checks that the derivative at the smallest initial value is positive, while the derivative at the largest initial value is negative. If this condition is not met, an error is returned. The 'hx' and 'hpx' vectors are passed to the 'make_z' function.

The 'make_z' function calculates the point of intersection between tangent lines to the log density function at successive abscissae x. The 'make_z' function takes as inputs the initial set of points, the left and right bounds, and the derivatives of the log density function at these points. The 'make_z' function uses equation (1) from Gilks and Wild (1992) to estimate the intersection points, using the lower and upper bounds as the first and last intersection points, respectively. These intersection points are used to compute the intervals over which each tangent line is the tightest upper bound of the log density function in the 'make_upper_bound' function. Though each tangent line is an upper bound over the domain of the entire log-density function, the 'z' vector allows us to identify the tangent line in each segment of the domain that is the tightest upper bound.

The 'make_z' function is tested with a plot of the 'z' values along with the associated values from the density function. Since the 'z' are just points from the density function, which is lognormal in this test case, we just plot two set of values together to see if they fall on the same line.

The 'make_upper_bound' and 'make_lower_bound' functions generate the piecewise-linear upper and lower bounds to the log density function over the entire domain. The 'make_upper_bound' function simply constructs the piecewise linear upper bound using a different tangent line at the intervals given by the 'z' vector (which is the output of 'make_z'). The output of the function ('upper_bound'), give parameters for each linear section of the upper bound, which are simply the point of tangency ('hx') and the slope of the tangent line ('hpx') at each of the intervals given by the 'z' vector. The 'make_lower_bound' function returns a piecewise-linear lower bound for log-density ('lower_bound') using secant lines between successive abscissa ('xx') and their associated log density values ('hx'). When an argument is out of the range of the abscissaes of x, the lower bound is set to $-\infty$. The remaining functions deal with sampling from the distribution using the upper and lower bounds created by these two functions.

The 'make_upper_bound' function is first tested by simply checking that all of the lower bounds we generate are less than upper bound we generate. Then, the 'make_upper_bound' function is tested by using a specific function ($f(x) = -x^2$) and plots both the upper and lower bounds. This second test allows us to visually inspect whether the upper bounds are constructed via tangent lines and the lower bounds are constructed via secant lines.

The 'sample_upper_bound' function takes as inputs 'x', 'hx', 'hpx', and 'z' and outputs a sample of 'm' points from the upper bound function. The points are sampled via the inverse CDF method, which is estimated analytically and implemented in the auxiliary function "inversecdf". First, the normalized integral of the upper bound function is estimated at each section of the piecewise function. Then 'm' points are sampled from a uniform distribution between zero and the 'Inormalize' value, where the 'Inormalize' value is the sum of the normalized integral of the upper bound taken at each interval. The interval each sample falls in is calculated ('sample_interval'), and the inverse CDF is computed (using the 'inversecdf' function) at those points. The 'inversecdf' function returns a set of candidate points ('cand') that are passed to the 'filter' function.

The 'sample_upper_bound' is tested

The 'filter' takes in the candidate points ('cand') generated by the 'sample_upper_bound' function. The sample values are accepted if the associated randomly generated samples from the uniform distribution lie between the lower and upper bounds (i.e., the "squeeze test"). These points from the 'cand' vector are entered into the 'accepted' vector until the "squeeze test" is failed. From that point on, the log density function is evaluated and the value is accepted if the value lies between the upper bound and the density function (these points from 'cand' are entered into the 'update' vector). All other points are rejected. The 'filter' function returns a list that contains a value for the last accepted point and a value for the first non-accepted point. If all of the points in the 'cand' vector are are accepted, the value for the first non-accepted point is simply "NA". Likewise, if no points from the 'cand' vector are accepted, the first accepted point value is simply “NA”. The ‘update_sample’ function adds the accepted sample into the final sample and checks whether the updated sample should be added to the final sample as well. In addition, the log-concavity of these points is checked by confirming that log of the function values falls between the upper and lower bounds.

The ‘filter’ function is tested by inputting a sample from the upper bound (based on random samples from the normal distribution), to confirm that ‘filter’ separates these candidate values into the ‘accepted’ and the ‘non-accepted’.

Finally, the ‘update_x’ function adds the points that are accepted by the ‘filter’ function into the abscissaes vector. The function also tests whether the updated sample is log-concave by checking whether the derivative at the updates points lie between their neighbouring points. An error is returned if the vector of derivatives at the updated points fail this test of log-concavity. The ‘while loop’ continues sampling, with the updated ‘x’ vector used in place of ‘x_init’ vector, until ‘n’ samples are accepted and the ‘ars’ function returns final accepted sample.

The ‘update_x’ test checks that an ‘hpx’ vector that is non-decreasing will generate an error.

Finally, we include a number of tests for the overall ‘ars’ function. The first test confirms that extra arguments can be passed to the function. The second test confirms that the function can handle a bounded distribution. The third test checks whether an chosen set of points are well-behaved. Finally, the fourth test checks whether the function will catch that the function is non-concave.

Contributions:

References:

Gilks, W. R. and Wild, P. (1992), “Adaptive Rejection Sampling for Gibbs Sampling,” *Applied Statistics*, Vol. 41, Issue 2, pp. 337-348.